**Blockchain Project (Part 1)**

Name: Muhammad Haziq Adli bin Zamzuri

Matric no: 1814981

File:

1. BankNegaraContract.sol

Smart contract where all specifications for cryptocurrency regulator’s basic role being written.

1. Migrations.sol

Initialization file (truffle init).

Code:

1. BankNegaraContract.sol

pragma solidity >=0.5.16 <0.8.0;

contract CryptocurrencyLaundererDetector {

// Initialize variable

uint private threshold = 10 \* 10\*\*18;

uint private maxBalance = 50 \* 10\*\*18;

address[] private empty;

address[] private transactors;

address[] private potentialLaunder;

uint[] private amountExceed;

uint private owner;

// Event for EVM logging

event OwnerSet(address indexed oldOwner, address indexed newOwner);

event Deposit(address indexed accountAddress, uint amount);

event Withdraw(address indexed accountAddress, uint amount);

mapping (address => uint256) public accountBalance;

address ownerBankNegara;

// Modifier to check if caller is owner

modifier BankNegara() {

require(msg.sender == ownerBankNegara);

\_;

}

// Set contract deployer as owner

constructor() public payable{

ownerBankNegara = msg.sender;

owner = 0;

}

// Set threshold

function setThreshold(uint thresholdVal) public BankNegara {

threshold = thresholdVal;

}

// View threshold

function getThreshold() public BankNegara view returns (uint) {

return threshold / (10\*\*18);

}

// Withdraw fund

function withdraw(uint withdraw\_amount) public {

// Limit withdrawal amount

require(address(this).balance >= withdraw\_amount \* 1 ether, "Insufficient Balance");

// Send the amount to the address that requested it

msg.sender.transfer(withdraw\_amount \* 1 ether);

transactors.push(msg.sender);

// Minus withdraw amount in account balance

accountBalance[msg.sender] -= withdraw\_amount\* 1 ether;

emit Withdraw(msg.sender, withdraw\_amount);

}

// Deposit fund

function deposit() public payable {

transactors.push(msg.sender);

accountBalance[msg.sender] += msg.value;

if(msg.value > threshold){

potentialLaunder.push(msg.sender);

amountExceed.push(msg.value);

}

emit Deposit(msg.sender, msg.value);

}

// Get account amount

function getAccAmount() public view returns (uint256) {

return accountBalance[msg.sender];

}

// Get balance in the account

function getBalance() public view returns (uint256) {

return address(this).balance;

}

// Get transactors

function getTransactors() public view returns (address[] memory) {

return transactors;

}

// Get transcation exceed max account balance

function getExceedTransactions() public BankNegara view returns (address[] memory, uint[] memory) {

return (potentialLaunder, amountExceed);

}

// Get potential launder

function getLaunder() public BankNegara view returns (address[] memory) {

if(address(this).balance > maxBalance)

return transactors;

else

return empty;

}

}

1. Migrations.sol

// SPDX-License-Identifier: MIT

pragma solidity >=0.4.22 <0.9.0;

contract Migrations {

address public owner = msg.sender;

uint public last\_completed\_migration;

modifier restricted() {

require(

msg.sender == owner,

"This function is restricted to the contract's owner"

);

\_;

}

function setCompleted(uint completed) public restricted {

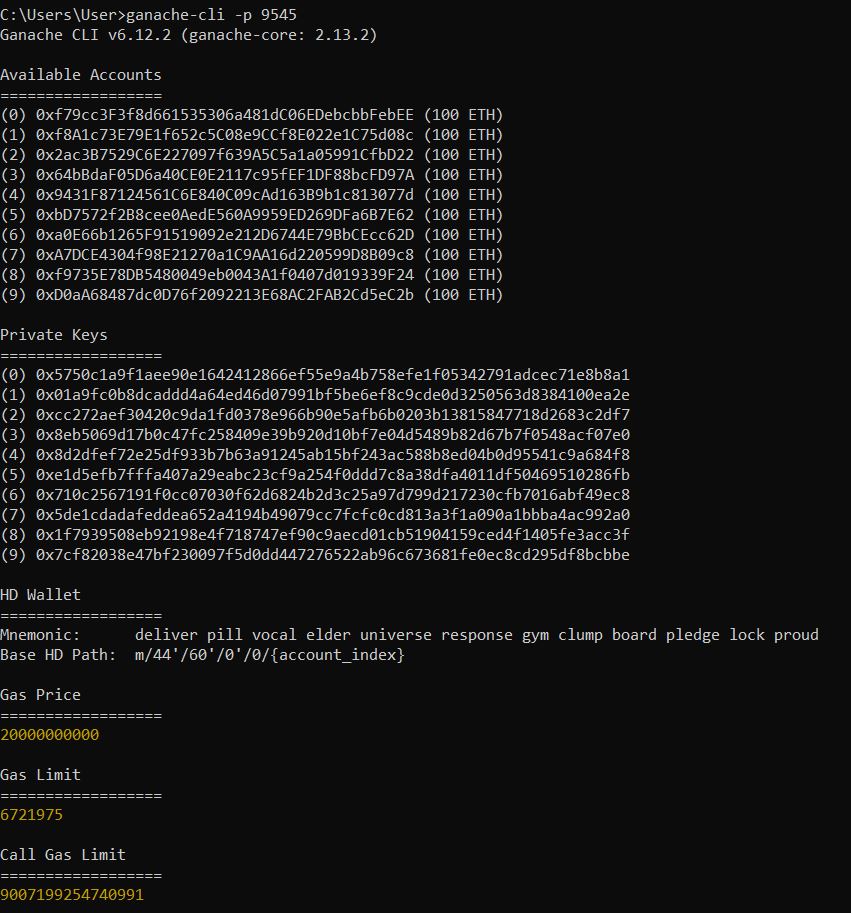
last\_completed\_migration = completed;

}

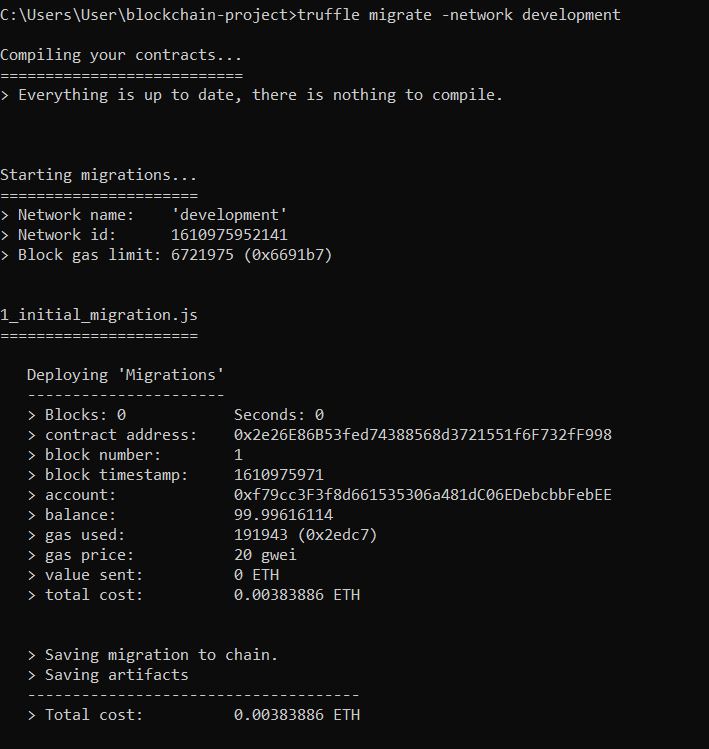
}

Snapshot:

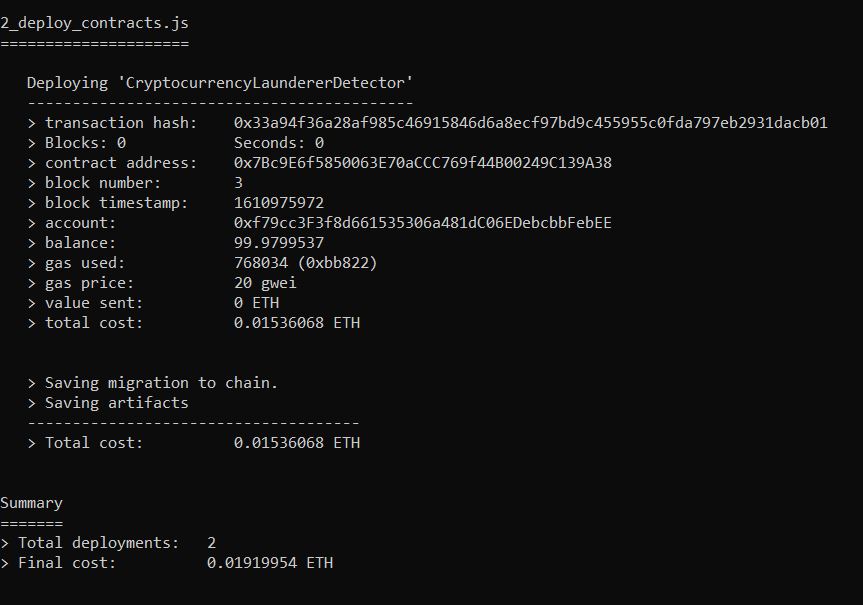
1. ganache\_local\_network



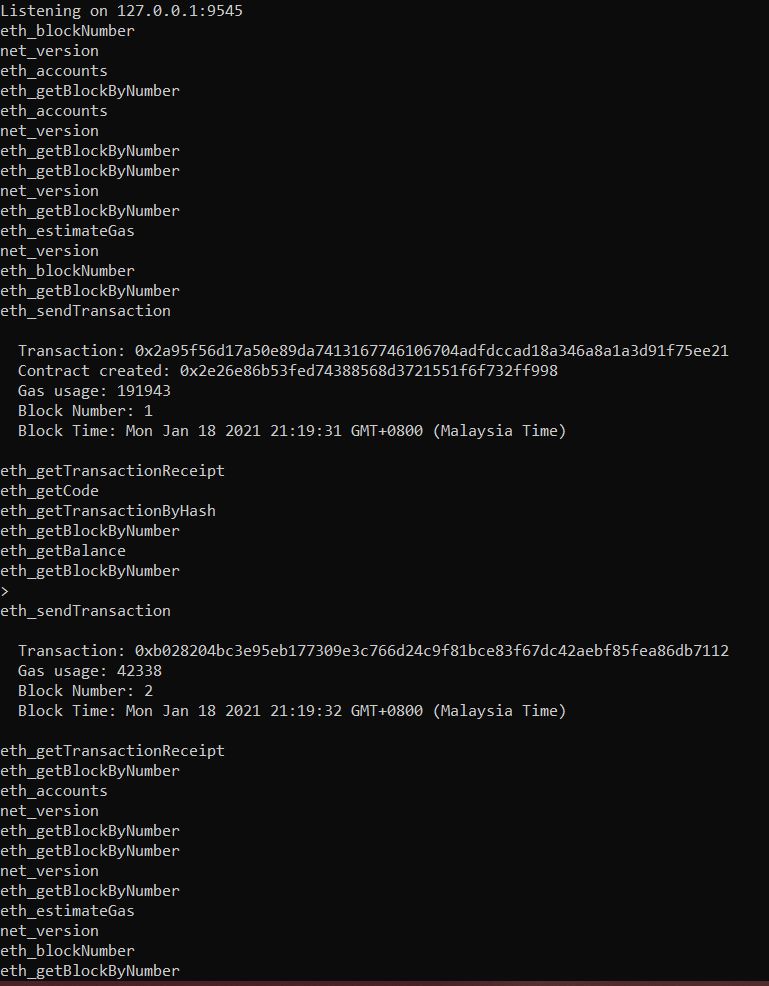
2. truffle\_migration\_1



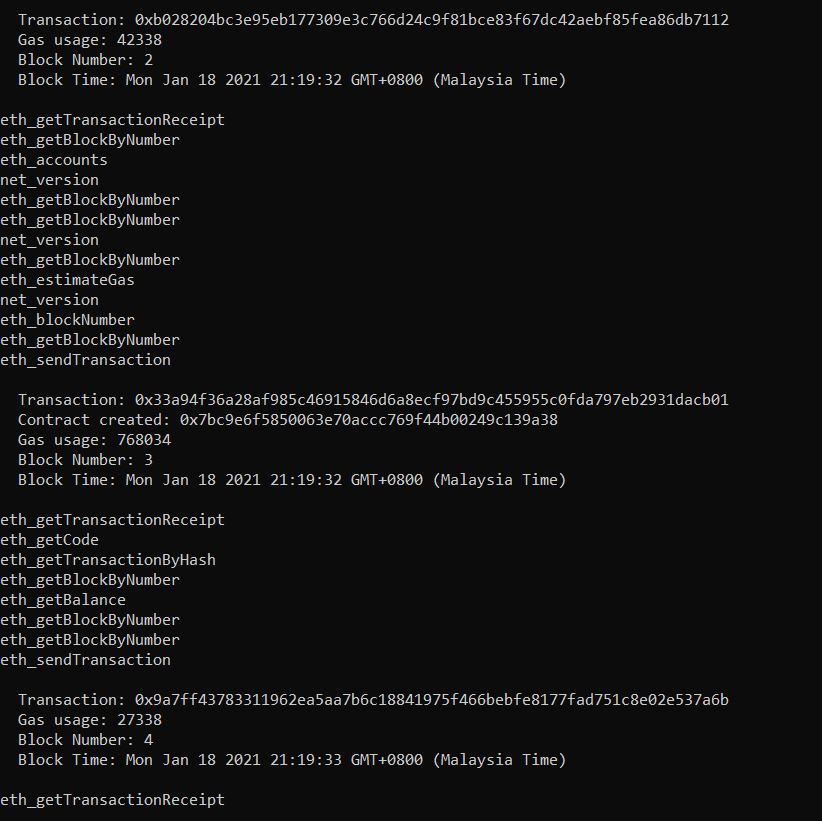
3. truffle\_migration\_2



4. ganache\_deploy\_1



5. ganache\_deploy\_2



6. truffle\_develop

